SQL & Python in Smart Water Management Assistant

Lambda Function

- นำเข้าข้อมูลปริมาณน้ำในอ่างเก็บน้ำ

```
import json
import boto3
import urllib.request

S3_BUCKET = 'water-manage-raw'
S3_KEY = 'reservoir/reservoir-data.ndjson'
API_URL = 'https://app.rid.go.th/reservoir/api/reservoir/public'

def lambda_handler(event, context):
    try:
        with urllib.request.urlopen(API_URL) as response:
            raw = response.read()
            decoded = raw.decode("utf-8-sig")
            data = json.loads(decoded)
            if 'data' not in data or not isinstance(data['data'], list):
                raise ValueError("Missing or invalid 'data' key")
            records = data['data']
            if not records:
                raise ValueError("No records found")
            print(f"Total records: {len(records)}")
            # แปลง JSON array → NDJSON string
            ndjson_str = "\n".join([json.dumps(record, ensure_ascii=False) for record in
records])
            # อัปโหลดลง S3
            s3 = boto3.client('s3')
            s3.put_object(
                Bucket=S3_BUCKET,
                Key=S3_KEY,
                Body=ndjson_str.encode("utf-8"),
                ContentType='application/x-ndjson'
            )
            return {
                'statusCode': 200,
                'body': f'Successfully uploaded NDJSON to s3://{S3_BUCKET}/{S3_KEY}'
            }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': f'Error occurred: {str(e)}'
```

```
        }
```

- นำเข้าข้อมูลปริมาณน้ำในเขื่อน

```
import json
import boto3
import urllib.request

S3_BUCKET = 'water-manage-raw'
S3_KEY = 'dam/dam-data.ndjson'
API_URL = 'https://app.rid.go.th/reservoir/api/dam/public'

def lambda_handler(event, context):
    try:
        with urllib.request.urlopen(API_URL) as response:
            raw = response.read()
            decoded = raw.decode("utf-8-sig")
            data = json.loads(decoded)
            if 'data' not in data or not isinstance(data['data'], list):
                raise ValueError("Missing or invalid 'data' key")
            records = data['data']
            if not records:
                raise ValueError("No records found")
            print(f"Total records: {len(records)}")
            # แปลง JSON array → NDJSON string
            ndjson_str = "\n".join([json.dumps(record, ensure_ascii=False) for record in
records])
            # อัปโหลดลง S3
            s3 = boto3.client('s3')
            s3.put_object(
                Bucket=S3_BUCKET,
                Key=S3_KEY,
                Body=ndjson_str.encode("utf-8"),
                ContentType='application/x-ndjson'
            )
            return {
                'statusCode': 200,
                'body': f'Successfully uploaded NDJSON to s3://{S3_BUCKET}/{S3_KEY}'
            }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': f'Error occurred: {str(e)}'
```

- นำเข้าข้อมูลปริมาณน้ำในเขื่อนรายวันในเดือนตุลาคม พ.ศ. 2567

```
import json
import boto3
import urllib.request

S3_BUCKET = 'water-manage-raw'
S3_KEY = 'October-2024/dam/10/01/01.ndjson' # เปลี่ยนfolder path และชื่อไฟล์ตามวันที่ที่ดึงข้อมูลมา 01-31
API_URL = 'https://app.rid.go.th/reservoir/api/dam/public/2024-10-01'# เปลี่ยนวันที่ที่ดึงข้อมูลมา 01-31

def lambda_handler(event, context):
    try:
        with urllib.request.urlopen(API_URL) as response:
            raw = response.read()
            decoded = raw.decode("utf-8-sig")
            data = json.loads(decoded)
            if 'data' not in data or not isinstance(data['data'], list):
                raise ValueError("Missing or invalid 'data' key")
            records = data['data']
            if not records:
                raise ValueError("No records found")
            print(f"Total records: {len(records)}")
            #แปลง JSON array → NDJSON string
            ndjson_str = "\n".join([json.dumps(record, ensure_ascii=False) for record in
records])
            # อัปโหลดลง S3
            s3 = boto3.client('s3')
            s3.put_object(
                Bucket=S3_BUCKET,
                Key=S3_KEY,
                Body=ndjson_str.encode("utf-8"),
                ContentType='application/x-ndjson'
            )
            return {
                'statusCode': 200,
                'body': f'Successfully uploaded NDJSON to s3://{S3_BUCKET}/{S3_KEY}'
            }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': f'Error occurred: {str(e)}'
        }
```

- นำเข้าข้อมูลปริมาณน้ำในอ่างเก็บน้ำย้อนหลังในเดือนตุลาคม พ.ศ. 2567

```
import json
import boto3
import urllib.request

S3_BUCKET = 'water-manage-raw'
S3_KEY = 'October-2024/reservoir/10/01/01.ndjson'# เปลี่ยนfolder path และชื่อไฟล์ตามวันที่ที่ดึงข้อมูลมา 01-31
API_URL = 'https://app.rid.go.th/reservoir/api/reservoir/public/2024-10-01'#เปลี่ยนวันที่ที่ดึงข้อมูลมา 01-31

def lambda_handler(event, context):
    try:
        with urllib.request.urlopen(API_URL) as response:
            raw = response.read()
            decoded = raw.decode("utf-8-sig")
            data = json.loads(decoded)
            if 'data' not in data or not isinstance(data['data'], list):
                raise ValueError("Missing or invalid 'data' key")
            records = data['data']
            if not records:
                raise ValueError("No records found")
            print(f"Total records: {len(records)}")
            # แปลง JSON array → NDJSON string
            ndjson_str = "\n".join([json.dumps(record, ensure_ascii=False) for record in records])
            # อัปโหลดลง S3
            s3 = boto3.client('s3')
            s3.put_object(
                Bucket=S3_BUCKET,
                Key=S3_KEY,
                Body=ndjson_str.encode("utf-8"),
                ContentType='application/x-ndjson'
            )
            return {
                'statusCode': 200,
                'body': f'Successfully uploaded NDJSON to s3://{S3_BUCKET}/{S3_KEY}'
            }

    except Exception as e:
        return {
            'statusCode': 500,
            'body': f'Error occurred: {str(e)}'
        }
```

Glue Job Script

- สำหรับอัปเดต Query Table ใน Amazon Athena ข้อมูลรายวันที่ Amazon Lambda ดึง API อ่างเก็บน้ำและเขื่อนและมีการเก็บเป็น folder รายวันใน S3 bucket

```
import boto3
import time
from datetime import datetime

ATHENA_DATABASE = 'watermanage'
ATHENA_OUTPUT = 's3://water-manage-raw/daily-api-auto/'
DAM_BASE_PATH = 's3://water-manage-raw/dam/dam-glue-job/'
RESERVOIR_BASE_PATH = 's3://water-manage-raw/reservoir/'

today = datetime.today().strftime('%Y-%m-%d')
dam_path = f"{DAM_BASE_PATH}dt={today}/"
res_path = f"{RESERVOIR_BASE_PATH}dt={today}/"

athena = boto3.client('athena')

def run_query(sql):
    response = athena.start_query_execution(
        QueryString=sql,
        QueryExecutionContext={'Database': ATHENA_DATABASE},
        ResultConfiguration={'OutputLocation': ATHENA_OUTPUT}
    )
    return response['QueryExecutionId']
def wait_for_query(qid):
    while True:
        state =
athena.get_query_execution(QueryExecutionId=qid)['QueryExecution']['Status']['State']
        if state in ['SUCCEEDED', 'FAILED', 'CANCELLED']:
            return state
        time.sleep(2)

# Job 1A: dam_data
print("[Job 1A.1] Dropping dam_data")
qid = run_query("DROP TABLE IF EXISTS dam_data;")
assert wait_for_query(qid) == 'SUCCEEDED'

print("[Job 1A.2] Creating dam_data")
qid = run_query("""
CREATE EXTERNAL TABLE dam_data (
  region STRING,
  dam ARRAY<STRUCT<
    id: STRING,
```

```
    name: STRING,
    storage: DOUBLE,
    dead_storage: DOUBLE,
    volume: DOUBLE,
    percent_storage: DOUBLE,
    inflow: DOUBLE,
    outflow: DOUBLE
  >>
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ('serialization.format' = '1')
LOCATION 's3://water-manage-raw/dam/api/'
TBLPROPERTIES ('has_encrypted_data'='false');
""")
assert wait_for_query(qid) == 'SUCCEEDED'
print("[Job 1A] dam_data created")

# Job 1B: reservoir_data
print("[Job 1B.1] Dropping reservoir_data")
qid = run_query("DROP TABLE IF EXISTS reservoir_data;")
assert wait_for_query(qid) == 'SUCCEEDED'

print("[Job 1B.2] Creating reservoir_data")
qid = run_query("""
CREATE EXTERNAL TABLE reservoir_data (
  region STRING,
  reservoirs ARRAY<STRUCT<
    id: STRING,
    name: STRING,
    owner: STRING,
    capacity: DOUBLE,
    storage: DOUBLE,
    active_storage: DOUBLE,
    dead_storage: DOUBLE,
    volume: DOUBLE,
    percent_storage: DOUBLE,
    inflow: DOUBLE,
    outflow: DOUBLE
  >>
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ('serialization.format' = '1')
LOCATION 's3://water-manage-raw/reservoir/api/'
TBLPROPERTIES ('has_encrypted_data'='false');
""")
```

```
assert wait_for_query(qid) == 'SUCCEEDED'
print("[Job 1B] reservoir_data created")

# Job 2A: dam_mapped_location
print("[Job 2A] Creating dam_mapped_location")
qid = run_query(f"""
CREATE TABLE dam_mapped_location
WITH (
  format = 'Parquet',
  external_location = '{dam_path}',
  write_compression = 'SNAPPY'
) AS
WITH flat_dam AS (
  SELECT
    region,
    r.name AS dam_name,
    r.id,
    r.storage,
    r.active_storage,
    r.dead_storage,
    r.volume,
    r.percent_storage,
    r.inflow,
    r.outflow
  FROM dam_data
  CROSS JOIN UNNEST(dam) AS t (r)
)
SELECT
  f.region,
  f.dam_name,
  l.subdistrict,
  l.district,
  l.province,
  f.id,
  f.storage,
  f.active_storage,
  f.dead_storage,
  f.volume,
  f.percent_storage,
  f.inflow,
  f.outflow
FROM flat_dam f
LEFT JOIN dam_location l
  ON TRIM(LOWER(f.dam_name)) = TRIM(LOWER(l.name));
""")
```

```
status = wait_for_query(qid)
if status != 'SUCCEEDED':
    print(f"Query failed: status={status}")
# Job 2B: reservoir_mapped_location
print("[Job 2B] Creating reservoir_mapped_location")
qid = run_query(f"""
CREATE TABLE reservoir_mapped_location
WITH (
  format = 'Parquet',
  external_location = '{res_path}',
  write_compression = 'SNAPPY'
) AS
WITH flat_reservoir AS (
  SELECT
    region,
    r.name AS reservoir_name,
    r.id,
    r.storage,
    r.dead_storage,
    r.volume,
    r.percent_storage,
    r.inflow,
    r.outflow
  FROM reservoir_data
  CROSS JOIN UNNEST(reservoirs) AS t (r)
)
SELECT
  f.region,
  f.reservoir_name,
  l.subdistrict,
  l.district,
  l.province,
  f.id,
  f.storage,
  f.dead_storage,
  f.volume,
  f.percent_storage,
  f.inflow,
  f.outflow
FROM flat_reservoir f
LEFT JOIN reservoir_location l
  ON TRIM(LOWER(f.reservoir_name)) = TRIM(LOWER(l.name));
""")
status = wait_for_query(qid)
if status != 'SUCCEEDED':
```

```
    print(f"Query failed: status={status}")
```

SQL บน Amazon Athena

- สร้างตาราง riskarea_processed ที่ cleaning ข้อมูล label ข้อมูล

```
CREATE TABLE riskarea_processed
WITH (
  format = 'PARQUET',
  external_location = 's3://water-manage-raw/risk-area/risk-area-label/',
  bucketed_by = ARRAY['location_key'],
  bucket_count = 1
) AS
SELECT
  CAST(month AS INT) AS month,
  TRIM(REPLACE(province_t, 'จ.', '')) AS province,
  TRIM(REPLACE(amphoe_t, 'อ.', '')) AS amphoe,
  TRIM(REPLACE(tambon_t, 'ต.', '')) AS tambon,
  CASE
    WHEN risk = 'เสี่ยงต่ำ' THEN 0
    WHEN risk = 'เสี่ยงปานกลาง' THEN 1
    WHEN risk = 'เสี่ยงสูง' THEN 2
    ELSE NULL
  END AS risk_label,
  CONCAT(
    TRIM(REPLACE(province_t, 'จ.', '')), '_',
    TRIM(REPLACE(amphoe_t, 'อ.', '')), '_',
    TRIM(REPLACE(tambon_t, 'ต.', ''))
  ) AS location_key
FROM riskarea
WHERE risk IS NOT NULL;
```

- สร้างตาราง Reservoir Table

```
CREATE EXTERNAL TABLE reservoir_data (
  region string,
  reservoir array<
    struct<
      id:string,
      name:string,
      storage:double,
      dead_storage:double,
      volume:double,
      percent_storage:double,
```

```
      inflow:double,
      outflow:double
    >
  >
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = '1'
)
LOCATION 's3://water-manage-raw/reservoir/api/'
TBLPROPERTIES ('has_encrypted_data'='false');
```

- สร้าง Table reservoir_location ที่มีข้อมูลของชื่ออ่างเก็บน้ำ และสถานที่ตั้ง

```
CREATE EXTERNAL TABLE reservoir_location (
  region STRING,
  name STRING,
  subdistrict STRING,
  district STRING,
  province STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar"     = "\""
)
LOCATION 's3://water-manage-raw/resevoir/location/'
TBLPROPERTIES (
  'has_encrypted_data'='false',
  'skip.header.line.count'='1'
);
```

- สร้างตาราง reservoir_with_location_mapped เพื่อ mapped ข้อมูลของตาราง reservoir_data และ reservoir_location เข้าด้วยกัน

```
CREATE TABLE reservoir_mapped_location
WITH (
  format = 'Parquet',
  external_location = 's3://water-manage-raw/reservoir/mapped-with-location/',
  write_compression = 'SNAPPY'
) AS

WITH flat_reservoir AS (
  SELECT
    region,
```

```
    r.name AS reservoir_name,
    r.id,
    r.storage,
    r.dead_storage,
    r.volume,
    r.percent_storage,
    r.inflow,
    r.outflow
  FROM reservoir_data
  CROSS JOIN UNNEST(reservoir) AS t (r)
)

SELECT
  f.region,
  f.reservoir_name,
  l.subdistrict,
  l.district,
  l.province,
  f.id,
  f.storage,
  f.dead_storage,
  f.volume,
  f.percent_storage,
  f.inflow,
  f.outflow
FROM flat_reservoir f
LEFT JOIN reservoir_location l
  ON TRIM(LOWER(f.reservoir_name)) = TRIM(LOWER(l.name))
```

- สร้างตาราง dam_data เพื่อดูข้อมูลของปริมาณน้ำในเขื่อน

```
CREATE EXTERNAL TABLE dam_data (
  region STRING,
  dam ARRAY<STRUCT<
    id: STRING,
    name: STRING,
    owner: STRING,
    capacity: DOUBLE,
    storage: DOUBLE,
    active_storage: DOUBLE,
    dead_storage: DOUBLE,
    volume: DOUBLE,
    percent_storage: DOUBLE,
    inflow: DOUBLE,
    outflow: DOUBLE
```

```
  >>
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = '1'
)
LOCATION 's3://water-manage-raw/dam/api/'
TBLPROPERTIES ('has_encrypted_data'='false');
```

- สร้างตาราง dam_location ที่มีชื่อเขื่อนและสถานที่ตั้ง

```
CREATE EXTERNAL TABLE dam_location (
  region STRING,
  name STRING,
  subdistrict STRING,
  district STRING,
  province STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar"     = "\""
)
LOCATION 's3://water-manage-raw/dam/location/'
TBLPROPERTIES (
  'has_encrypted_data'='false',
  'skip.header.line.count'='1'
);
```

- สร้างตาราง dam_with_location_mapped เพื่อ mapped ข้อมูลของตาราง dam_data และ dam_location เข้าด้วยกัน

```
CREATE TABLE dam_with_location_mapped
WITH (
  format = 'Parquet',
  external_location = 's3://water-manage-raw/dam/dam_with_location_mapped/',
  write_compression = 'SNAPPY'
) AS

WITH flat_dam AS (
  SELECT
    region,
    r.name AS dam_name,
    r.id,
    r.storage,
```

```
    r.active_storage,
    r.dead_storage,
    r.volume,
    r.percent_storage,
    r.inflow,
    r.outflow
  FROM dam_data_manual
  CROSS JOIN UNNEST(dam) AS t (r)
)

SELECT
  f.region,
  f.dam_name,
  l.province,
  l.district,
  l.subdistrict,
  f.id,
  f.storage,
  f.active_storage,
  f.dead_storage,
  f.volume,
  f.percent_storage,
  f.inflow,
  f.outflow
FROM flat_dam f
LEFT JOIN dam_location l
  ON TRIM(LOWER(f.dam_name)) = TRIM(LOWER(l.name))
```

- สร้างตาราง rainfall_october

```
CREATE EXTERNAL TABLE rainfall_october (
  station_code STRING,
  measure_datetime STRING,
  rainfall_daily_raw  STRING,
  quality_flag STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  'separatorChar' = ',',
  'quoteChar' = '"'
)
LOCATION 's3://water-manage-raw/October-2024/rainfall/raw'
TBLPROPERTIES (
  'has_encrypted_data'='false',
  'skip.header.line.count'='1'
```

```
);
```

- เรียกดูข้อมูลมูลบาง Field

```
SELECT
  measure_datetime,
  rainfall_daily_raw
FROM rainfall_october
LIMIT 20;
```

- สร้างตาราง rainfall_avg

```
CREATE TABLE rainfall_avg
WITH (
  format = 'PARQUET',
  external_location = 's3://water-manage-raw/October-2024/rainfall/avg-rainfall',
  bucketed_by = ARRAY['station_code'],
  bucket_count = 1
) AS
SELECT
  station_code,
  AVG(TRY_CAST(rainfall_daily_raw AS DOUBLE)) AS avg_rainfall_mm
FROM rainfall_october
WHERE TRY_CAST(rainfall_daily_raw AS DOUBLE) IS NOT NULL
GROUP BY station_code
ORDER BY avg_rainfall_mm DESC;
```

- สร้างตาราง rainfall_station

```
CREATE EXTERNAL TABLE rainfall_station (
  station_code STRING,
  station_name STRING,
  latitude STRING,
  longitude STRING,
  basin_name STRING,
  sub_basin_name STRING,
  tambon_name STRING,
  amphoe_name STRING,
  province_name STRING,
  station_type_name STRING,
  region_name STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
```

```
  "quoteChar" = "\""
)
LOCATION 's3://water-manage-raw/October-2024/rainfall/location'
TBLPROPERTIES (
  'skip.header.line.count'='1',
  'has_encrypted_data'='false'
);
```

- สร้างตาราง rain_avg_with_location เพื่อ mapped ข้อมูลของตาราง rainfall_avg และ rainfall_location เข้าด้วยกัน

```
CREATE TABLE rainfall_avg_with_location
WITH (
  format = 'PARQUET',
  external_location = 's3://water-manage-raw/October-2024/rainfall/avg-with-location'
) AS
SELECT
  a.station_code,
  a.avg_rainfall_mm,
  m.station_name,
  m.tambon_name,
  m.amphoe_name,
  m.province_name,
  m.station_type_name,
  m.region_name,
  m.basin_name
FROM rainfall_avg a
LEFT JOIN rainfall_station m
  ON LOWER(REPLACE(a.station_code, ' ', '')) = LOWER(REPLACE(m.station_code, ' ', ''));
```

- การรวม reservoir in oct ทั้งหมด 31 วันโดยเริ่มจากการสร้างตารางของทุกวันก่อน

```
CREATE EXTERNAL TABLE reservoir_oct_31 (
  region STRING,
  reservoir ARRAY<STRUCT<
      id:string,
      name:string,
      storage:double,
      dead_storage:double,
      volume:double,
      percent_storage:double,
      inflow:double,
      outflow:double
  >>
)
```

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = '1'
)
LOCATION 's3://water-manage-raw/October-2024/reservoir/10/01'#เปลี่ยนโฟลเดอร์วันที่ที่ดึงข้อมูลมา 01-31
TBLPROPERTIES (
'has_encrypted_data'='false','classification'='json'
);
```

```
CREATE TABLE Oct_31
WITH (
  format = 'Parquet',
  external_location = 's3://water-manage-raw/October-2024/reservoir/mapping/31/', #เปลี่ยนโฟลเดอร์วันที่ที่ดึงข้อมูลมา 01-31
  write_compression = 'SNAPPY'
) AS
WITH flat_reservoir AS (
  SELECT
    region,
    r.name AS reservoir_name,
    r.id,
    r.storage,
    r.dead_storage,
    r.volume,
    r.percent_storage,
    r.inflow,
    r.outflow
  FROM reservoir_oct_31
  CROSS JOIN UNNEST(reservoir) AS t (r)
)
SELECT
  f.region,
  f.reservoir_name,
  l.province,
  l.district,
  l.subdistrict,
  f.id,
  f.storage,
  f.dead_storage,
  f.volume,
  f.percent_storage,
  f.inflow,
  f.outflow
FROM flat_reservoir f
```

```
LEFT JOIN reservoir_location l
  ON TRIM(LOWER(f.reservoir_name)) = TRIM(LOWER(l.name))
```

- สร้างตารางรวมข้อมูลทั้ง 31 วัน

```
CREATE TABLE reservoir_oct_2024
WITH (
  format = 'PARQUET',
  external_location = 's3://water-manage-raw/October-2024/reservoir/joint/'
) AS

SELECT * FROM reservoir.oct_01
UNION ALL
SELECT * FROM reservoir.oct_02
UNION ALL
SELECT * FROM reservoir.oct_03
UNION ALL
SELECT * FROM reservoir.oct_04
UNION ALL
SELECT * FROM reservoir.oct_05
UNION ALL
SELECT * FROM reservoir.oct_06
UNION ALL
SELECT * FROM reservoir.oct_07
UNION ALL
SELECT * FROM reservoir.oct_08
UNION ALL
SELECT * FROM reservoir.oct_09
UNION ALL
SELECT * FROM reservoir.oct_10
UNION ALL
SELECT * FROM reservoir.oct_11
UNION ALL
SELECT * FROM reservoir.oct_12
UNION ALL
SELECT * FROM reservoir.oct_13
UNION ALL
SELECT * FROM reservoir.oct_14
UNION ALL
SELECT * FROM reservoir.oct_15
UNION ALL
SELECT * FROM reservoir.oct_16
UNION ALL
SELECT * FROM reservoir.oct_17
```

```
UNION ALL
SELECT * FROM reservoir.oct_18
UNION ALL
SELECT * FROM reservoir.oct_19
UNION ALL
SELECT * FROM reservoir.oct_20
UNION ALL
SELECT * FROM reservoir.oct_21
UNION ALL
SELECT * FROM reservoir.oct_22
UNION ALL
SELECT * FROM reservoir.oct_23
UNION ALL
SELECT * FROM reservoir.oct_24
UNION ALL
SELECT * FROM reservoir.oct_25
UNION ALL
SELECT * FROM reservoir.oct_26
UNION ALL
SELECT * FROM reservoir.oct_27
UNION ALL
SELECT * FROM reservoir.oct_28
UNION ALL
SELECT * FROM reservoir.oct_29
UNION ALL
SELECT * FROM reservoir.oct_30
UNION ALL
SELECT * FROM reservoir.oct_31;
```

- หาค่าเฉลี่ยปริมาณน้ำในอ่างเก็บน้ำในเดือน

```
CREATE TABLE reservoir_avg_oct_2024
WITH (
  format = 'PARQUET',
  external_location = 's3://water-manage-raw/October-2024/reservoir/preprocessed/'
) AS
SELECT
  id,
  MIN(reservoir_name) AS reservoir_name,
  MIN(province) AS province,
  MIN(district) AS district,
  MIN(subdistrict) AS subdistrict,
  AVG(TRY_CAST(volume AS DOUBLE)) AS avg_volume,
  AVG(TRY_CAST(percent_storage AS DOUBLE)) AS avg_percent_storage,
```

```
  AVG(TRY_CAST(inflow AS DOUBLE)) AS avg_inflow,
  AVG(TRY_CAST(outflow AS DOUBLE)) AS avg_outflow
FROM reservoir_oct_2024
WHERE id IS NOT NULL
GROUP BY id;
```

- การรวม dam in oct ทั้งหมด 31 วันโดยเริ่มจากการสร้างตารางของทุกวันก่อน

```
CREATE EXTERNAL TABLE dam_oct_31 (
  region STRING,
  dam ARRAY<STRUCT<
    id: STRING,
    name: STRING,
    owner: STRING,
    capacity: DOUBLE,
    storage: DOUBLE,
    active_storage: DOUBLE,
    dead_storage: DOUBLE,
    volume: DOUBLE,
    percent_storage: DOUBLE,
    inflow: DOUBLE,
    outflow: DOUBLE
  >>
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = '1'
)
LOCATION 's3://water-manage-raw/October-2024/dam/10/31/'
TBLPROPERTIES (
'has_encrypted_data'='false','classification'='json'
);
```

- Mapping dam_oct_xx และ dam_location

```
CREATE TABLE Oct_31
WITH (
  format = 'Parquet',
  external_location = 's3://water-manage-raw-data/October-2024/dam/mapped-location/31/',
  write_compression = 'SNAPPY'
) AS

WITH flat_dam AS (
  SELECT
    region,
```

```
    r.name AS dam_name,
    r.id,
    r.storage,
    r.active_storage,
    r.dead_storage,
    r.volume,
    r.percent_storage,
    r.inflow,
    r.outflow
  FROM dam_oct_31
  CROSS JOIN UNNEST(dam) AS t (r)
)

SELECT
  f.region,
  f.dam_name,
  l.province,
  l.district,
  l.subdistrict,
  f.id,
  f.storage,
  f.active_storage,
  f.dead_storage,
  f.volume,
  f.percent_storage,
  f.inflow,
  f.outflow
FROM flat_dam f
LEFT JOIN dam_location l
  ON TRIM(LOWER(f.dam_name)) = TRIM(LOWER(l.name))
```

- สร้างตารางรวมทั้ง 31 วัน

```
CREATE TABLE dam_oct_2024
WITH (
  format = 'PARQUET',
  external_location = 's3://water-manage-raw/October-2024/dam/joint/'
) AS

SELECT * FROM dam.oct_01
UNION ALL
SELECT * FROM dam.oct_02
UNION ALL
SELECT * FROM dam.oct_03
```

```
UNION ALL
SELECT * FROM dam.oct_04
UNION ALL
SELECT * FROM dam.oct_05
UNION ALL
SELECT * FROM dam.oct_06
UNION ALL
SELECT * FROM dam.oct_07
UNION ALL
SELECT * FROM dam.oct_08
UNION ALL
SELECT * FROM dam.oct_09
UNION ALL
SELECT * FROM dam.oct_10
UNION ALL
SELECT * FROM dam.oct_11
UNION ALL
SELECT * FROM dam.oct_12
UNION ALL
SELECT * FROM dam.oct_13
UNION ALL
SELECT * FROM dam.oct_14
UNION ALL
SELECT * FROM dam.oct_15
UNION ALL
SELECT * FROM dam.oct_16
UNION ALL
SELECT * FROM dam.oct_17
UNION ALL
SELECT * FROM dam.oct_18
UNION ALL
SELECT * FROM dam.oct_19
UNION ALL
SELECT * FROM dam.oct_20
UNION ALL
SELECT * FROM dam.oct_21
UNION ALL
SELECT * FROM dam.oct_22
UNION ALL
SELECT * FROM dam.oct_23
UNION ALL
SELECT * FROM dam.oct_24
UNION ALL
SELECT * FROM dam.oct_25
UNION ALL
```

```
SELECT * FROM dam.oct_26
UNION ALL
SELECT * FROM dam.oct_27
UNION ALL
SELECT * FROM dam.oct_28
UNION ALL
SELECT * FROM dam.oct_29
UNION ALL
SELECT * FROM dam.oct_30
UNION ALL
SELECT * FROM dam.oct_31;
```

```
CREATE TABLE dam_avg_oct_2024
WITH (
  format = 'PARQUET',
  external_location = 's3://water-manage-raw/October-2-24/dam/processed'
) AS
SELECT
  id,
  MIN(region) AS region,
  MIN(dam_name) AS dam_name,
  MIN(province) AS province,
  MIN(district) AS district,
  MIN(subdistrict) AS subdistrict,
  AVG(TRY_CAST(active_storage AS DOUBLE)) AS avg_active_storage,
  AVG(TRY_CAST(volume AS DOUBLE)) AS avg_volume,
  AVG(TRY_CAST(percent_storage AS DOUBLE)) AS avg_percent_storage,
  AVG(TRY_CAST(inflow AS DOUBLE)) AS avg_inflow,
  AVG(TRY_CAST(outflow AS DOUBLE)) AS avg_outflow
FROM dam_oct_2024
WHERE id IS NOT NULL
GROUP BY id;
```

- เริ่มสร้าง master file ของเดือนตุลาคม พ.ศ. เพื่อใช้วิเคราะห์ข้อมูลใน Amazon SageMaker

```
CREATE EXTERNAL TABLE risk_area (
    month INT,
    province STRING,
    amphoe STRING,
    tambon STRING,
    risk_label INT,
```

```
    location_key STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    'separatorChar' = ',',
    'quoteChar' = '"'
)
LOCATION 's3://water-manage-raw/risk-area/risk-area-label/'
TBLPROPERTIES ('skip.header.line.count'='1');
```

```
CREATE EXTERNAL TABLE reservoir_avg (
    id STRING,
    reservoir_name STRING,
    tambon STRING,
    amphoe STRING,
    province STRING,
    avg_volume STRING,
    avg_percent_storage STRING,
    avg_inflow STRING,
    avg_outflow STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    'separatorChar' = ',',
    'quoteChar' = '"'
)
LOCATION 's3://water-manage-raw/October-2024/reservoir/preprocessed/'
TBLPROPERTIES ('skip.header.line.count'='1');
```

- แปลงช่อง string บางช่องที่นำเข้ามาเป็น string ให้กลายเป็น int

```
CREATE TABLE reservoir_avg_int AS
SELECT
    id,
    reservoir_name,
    tambon,
    amphoe,
    province,
    CAST(NULLIF(avg_volume, '') AS DOUBLE) AS avg_volume,
    CAST(NULLIF(avg_percent_storage, '') AS DOUBLE) AS avg_percent_storage,
    CAST(NULLIF(avg_inflow, '') AS DOUBLE) AS avg_inflow,
    CAST(NULLIF(avg_outflow, '') AS DOUBLE) AS avg_outflow
FROM reservoir_avg;
```

- เริ่ม joint ข้อมูลและทำตารางรีวิว

```
CREATE OR REPLACE VIEW reservoir_avg_oct AS
SELECT
    id,
    reservoir_name,
    TRIM(LOWER(REPLACE(province, 'จ.', ''))) AS province,
    TRIM(LOWER(REPLACE(amphoe, 'อ.', ''))) AS district,
    TRIM(LOWER(REPLACE(tambon, 'ต.', ''))) AS subdistrict,

    CAST(NULLIF(CAST(avg_volume AS VARCHAR), '') AS DOUBLE) AS avg_volume,
    CAST(NULLIF(CAST(avg_percent_storage AS VARCHAR), '') AS DOUBLE) AS avg_percent_storage,
    CAST(NULLIF(CAST(avg_inflow AS VARCHAR), '') AS DOUBLE) AS avg_inflow,
    CAST(NULLIF(CAST(avg_outflow AS VARCHAR), '') AS DOUBLE) AS avg_outflow,

    -- สร้างjoin key
    TRIM(LOWER(REPLACE(tambon, 'ต.', ''))) || '_' ||
    TRIM(LOWER(REPLACE(amphoe, 'อ.', ''))) || '_' ||
    TRIM(LOWER(REPLACE(province, 'จ.', ''))) AS join_key
FROM reservoir_avg_int;
```

```
CREATE OR REPLACE VIEW risk_with_reservoir AS
SELECT
    r.*,
    res.reservoir_name AS res_reservoir_name,
    res.avg_volume AS res_avg_volume,
    res.avg_percent_storage AS res_avg_percent_storage,
    res.avg_inflow AS res_avg_inflow,
    res.avg_outflow AS res_avg_outflow
FROM risk_area_master r
LEFT JOIN reservoir_avg_oct res
  ON r.join_key = res.join_key;
```

```
CREATE EXTERNAL TABLE dam_avg (
    id STRING,
    region STRING,
    dam_name STRING,
    tambon STRING,
    amphoe STRING,
    province STRING,
    avg_active_storage STRING,
    avg_volume STRING,
    avg_percent_storage STRING,
    avg_inflow STRING,
```

```
    avg_outflow STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    'separatorChar' = ',',
    'quoteChar' = '"'
)
LOCATION 's3://water-manage-raw/October-2024/dam/processed'
TBLPROPERTIES ('skip.header.line.count'='1');
```

```
CREATE TABLE dam_avg_int AS
SELECT
    id,
    dam_name,
    tambon,
    amphoe,
    province,
    CAST(NULLIF(avg_active_storage, '') AS DOUBLE) AS avg_active_storage,
    CAST(NULLIF(avg_volume, '') AS DOUBLE) AS avg_volume,
    CAST(NULLIF(avg_percent_storage, '') AS DOUBLE) AS avg_percent_storage,
    CAST(NULLIF(avg_inflow, '') AS DOUBLE) AS avg_inflow,
    CAST(NULLIF(avg_outflow, '') AS DOUBLE) AS avg_outflow
FROM dam_avg;
```

```
CREATE OR REPLACE VIEW dam_avg_oct AS
SELECT
    id,
    dam_name,
    TRIM(LOWER(REPLACE(province, 'จ.', ''))) AS province,
    TRIM(LOWER(REPLACE(amphoe, 'อ.', ''))) AS district,
    TRIM(LOWER(REPLACE(tambon, 'ต.', ''))) AS subdistrict,

    CAST(NULLIF(CAST(avg_active_storage AS VARCHAR), '') AS DOUBLE) AS avg_active_storage,
    CAST(NULLIF(CAST(avg_volume AS VARCHAR), '') AS DOUBLE) AS avg_volume,
    CAST(NULLIF(CAST(avg_percent_storage AS VARCHAR), '') AS DOUBLE) AS avg_percent_storage,
    CAST(NULLIF(CAST(avg_inflow AS VARCHAR), '') AS DOUBLE) AS avg_inflow,
    CAST(NULLIF(CAST(avg_outflow AS VARCHAR), '') AS DOUBLE) AS avg_outflow,

    -- สร้างjoin key
    TRIM(LOWER(REPLACE(tambon, 'ต.', ''))) || '_' ||
    TRIM(LOWER(REPLACE(amphoe, 'อ.', ''))) || '_' ||
    TRIM(LOWER(REPLACE(province, 'จ.', ''))) AS join_key
FROM dam_avg_int;
```

```
CREATE OR REPLACE VIEW risk_with_reservoir_dam AS
SELECT
    r.*,
    res.dam_name AS res_dam_name,
    res.avg_active_storage AS res_dam_avg_active_storage,
    res.avg_volume AS res_dam_avg_volume,
    res.avg_percent_storage AS res_dam_avg_percent_storage,
    res.avg_inflow AS res_dam_avg_inflow,
    res.avg_outflow AS res_dam_avg_outflow
FROM risk_with_reservoir r
LEFT JOIN dam_avg_oct res
  ON r.join_key = res.join_key;
```

```
CREATE EXTERNAL TABLE rainfall_avg (
    station_code STRING,
    avg_rainfall_mm DOUBLE,
    station_name STRING,
    tambon_name STRING,
    amphoe_name STRING,
    province_name STRING,
    station_type_name STRING,
    region_name STRING,
    basin_name STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    'separatorChar' = ',',
    'quoteChar' = '"'
)
LOCATION 's3://water-manage-raw/October-2024/rainfall/avg-with-location'
TBLPROPERTIES ('skip.header.line.count'='1');
```

```
CREATE OR REPLACE VIEW rainfall_avg_oct AS
SELECT
    station_code,
    station_name,
    station_type_name,
    TRIM(LOWER(REPLACE(province_name, 'จ.', ''))) AS province,
    TRIM(LOWER(REPLACE(amphoe_name, 'อ.', ''))) AS district,
    TRIM(LOWER(REPLACE(tambon_name, 'ต.', ''))) AS subdistrict,

    CAST(NULLIF(CAST(avg_rainfall_mm AS VARCHAR), '') AS DOUBLE) AS avg_rainfall_mm,

    -- สร้างjoin key
```

```
    TRIM(LOWER(REPLACE(tambon_name, 'ต.', ''))) || '_' ||
    TRIM(LOWER(REPLACE(amphoe_name, 'อ.', ''))) || '_' ||
    TRIM(LOWER(REPLACE(province_name, 'จ.', ''))) AS join_key
FROM rainfall_avg;
```

- Master file ที่นำไปในการวิเคราะห์

```
CREATE EXTERNAL TABLE IF NOT EXISTS `watermanage`.`all_oct_2024` (
  `month` int,
  `province` string,
  `district` string,
  `subdistrict` string,
  `risk_label` int,
  `location_key` string,
  `join_key` string,
  `res_reservoir_name` string,
  `res_avg_volume` double,
  `res_avg_percent_storage` double,
  `res_avg_inflow` double,
  `res_avg_outflow` double,
  `res_dam_name` string,
  `res_dam_avg_volume` double,
  `res_dam_avg_percent_storage` double,
  `res_dam_avg_inflow` double,
  `res_dam_avg_outflow` double,
  `res_rainfall_station_name` string,
  `res_station_type_name` string,
  `res_avg_rainfall_mm` double
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES ('field.delim' = ',')
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://water-manage-raw/October-2024/'
TBLPROPERTIES ('classification' = 'csv','skip.header.line.count'='1');
```

Amazon SageMaker

- ทดสอบใช้ K-cluster ในการจัดกลุ่มพื้นที่เสี่ยงน้ำท่วม

```
# STEP 1: Import Libraries
import boto3
import pandas as pd
import numpy as np
from io import StringIO
from sklearn.impute import SimpleImputer
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import silhouette_score
from scipy.stats import mode
import matplotlib.pyplot as plt
from IPython.display import display
# STEP 2: S3 Configuration
input_bucket = 'water-manage-raw'
input_key = 'October-2024/risk_with_all_oct.csv'
output_bucket = 'water-manage-processed'
output_key = 'k-cluster/kmeans_results.csv'
region = 'us-east-1'
s3 = boto3.client('s3', region_name=region)
# STEP 3: Load data from S3
response = s3.get_object(Bucket=input_bucket, Key=input_key)
df = pd.read_csv(response['Body'])

# STEP 4 (ปรับปรุง): Fill Missing แบบ Mapping ก่อน Scaling
features = [
    'res_avg_volume', 'res_avg_percent_storage', 'res_avg_inflow', 'res_avg_outflow',
    'res_dam_avg_volume', 'res_dam_avg_percent_storage', 'res_dam_avg_inflow',
'res_dam_avg_outflow',
    'res_avg_rainfall_mm'
]

# กำหนดลิสต์ field ที่จะเติมจากตำบล -> อำเภอ -> จังหวัด
res_fields = ['res_avg_volume', 'res_avg_percent_storage', 'res_avg_inflow', 'res_avg_outflow']
dam_fields = ['res_dam_avg_volume', 'res_dam_avg_percent_storage', 'res_dam_avg_inflow',
'res_dam_avg_outflow']
rain_fields = ['res_avg_rainfall_mm']

# Copy ข้อมูลสำหรับ fill missing
df_filled = df.copy()

# เติม RESERVOIR ระดับ ตำบล -> อำเภอ -> จังหวัด
for col in res_fields:
    df_filled[col] = df_filled.groupby(['subdistrict'])[col].transform(lambda x:
x.fillna(x.mean()))
    df_filled[col] = df_filled.groupby(['district'])[col].transform(lambda x: x.fillna(x.mean()))
    df_filled[col] = df_filled.groupby(['province'])[col].transform(lambda x: x.fillna(x.mean()))

#เติม DAM ระดับ อำเภอ -> จังหวัด
for col in dam_fields:
```

```
    df_filled[col] = df_filled.groupby(['district'])[col].transform(lambda x: x.fillna(x.mean()))
    df_filled[col] = df_filled.groupby(['province'])[col].transform(lambda x: x.fillna(x.mean()))

#เติม RAINFALL ระดับ อำเภอ
for col in rain_fields:
    df_filled[col] = df_filled.groupby(['district'])[col].transform(lambda x: x.fillna(x.mean()))

# เตรียม X, y ใหม่หลังเติมข้อมูล
X = df_filled[features]
y_true = df_filled['risk_label'] if 'risk_label' in df_filled.columns else None

# Scaling หลังจาก Fill
X_imputed = SimpleImputer(strategy='mean').fit_transform(X)
X_scaled = StandardScaler().fit_transform(X_imputed)
# STEP 5 (optional): Elbow Method
inertia = []
K_range = range(1, 7)
for k in K_range:
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X_scaled)
    inertia.append(km.inertia_)

plt.plot(K_range, inertia, marker='o')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal k")
plt.grid(True)
plt.show()
#  STEP 6: Run K-Means with chosen k
k = 5  # <-- เปลี่ยนได้ตามผล elbow
kmeans = KMeans(n_clusters=k, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled)

# STEP 7: PCA for visualization
pca = PCA(n_components=2)
df[['pca1', 'pca2']] = pca.fit_transform(X_scaled)
# STEP 8: Evaluate clustering (only if risk_label is available)
if y_true is not None:
    def map_clusters_to_labels(clusters, true_labels):
        labels = np.zeros_like(clusters)
        for i in np.unique(clusters):
            mask = (clusters == i)
            labels[mask] = mode(true_labels[mask], keepdims=True)[0]
        return labels
```

```
    mapped_preds = map_clusters_to_labels(df['cluster'].values, y_true.values)

    print("\n📊 Classification Report for K-Means:")
    print(classification_report(y_true, mapped_preds))

    cm = confusion_matrix(y_true, mapped_preds)
    ConfusionMatrixDisplay(cm).plot(cmap='Blues')
    plt.title("Confusion Matrix: K-Means vs True Risk Label")
    plt.show()

    # Visualize PCA clusters
    plt.figure(figsize=(8, 6))
    for c in np.unique(df['cluster']):
        subset = df[df['cluster'] == c]
        plt.scatter(subset['pca1'], subset['pca2'], label=f'Cluster {c}', alpha=0.6)
    plt.xlabel("PCA1")
    plt.ylabel("PCA2")
    plt.title("PCA Cluster Scatter Plot")
    plt.legend()
    plt.grid()
    plt.show()
# STEP 9: Export results to S3
csv_buffer = StringIO()
df.to_csv(csv_buffer, index=False)
s3.put_object(Bucket=output_bucket, Key=output_key, Body=csv_buffer.getvalue())
print(f"✅ Results saved to s3://{output_bucket}/{output_key}")
```